HW15 – AOP

goteam application

Aspect-Oriented Programming (AOP) is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns. It does this by adding additional behaviour to existing code (an advice) without modifying the code itself, instead separately specifying which code is modified via a “pointcut” specification. This approach is in contrast to traditional object-oriented programming, which typically ties behaviour closely to the object to which it pertains.

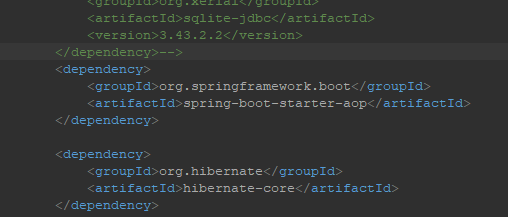
In the context of my Spring Boot application, I am using AOP to intercept CRUD (Create, Read, Update, Delete) operations in your service layer. This is typically done to add common functionalities like logging, which is a cross-cutting concern and should ideally be separated from the main business logic.

For example, by using AOP, we can log every time a CRUD operation is performed without having to add logging statements to each method. This not only keeps your service methods clean and focused solely on business logic but also allows you to manage logging in a centralized way.

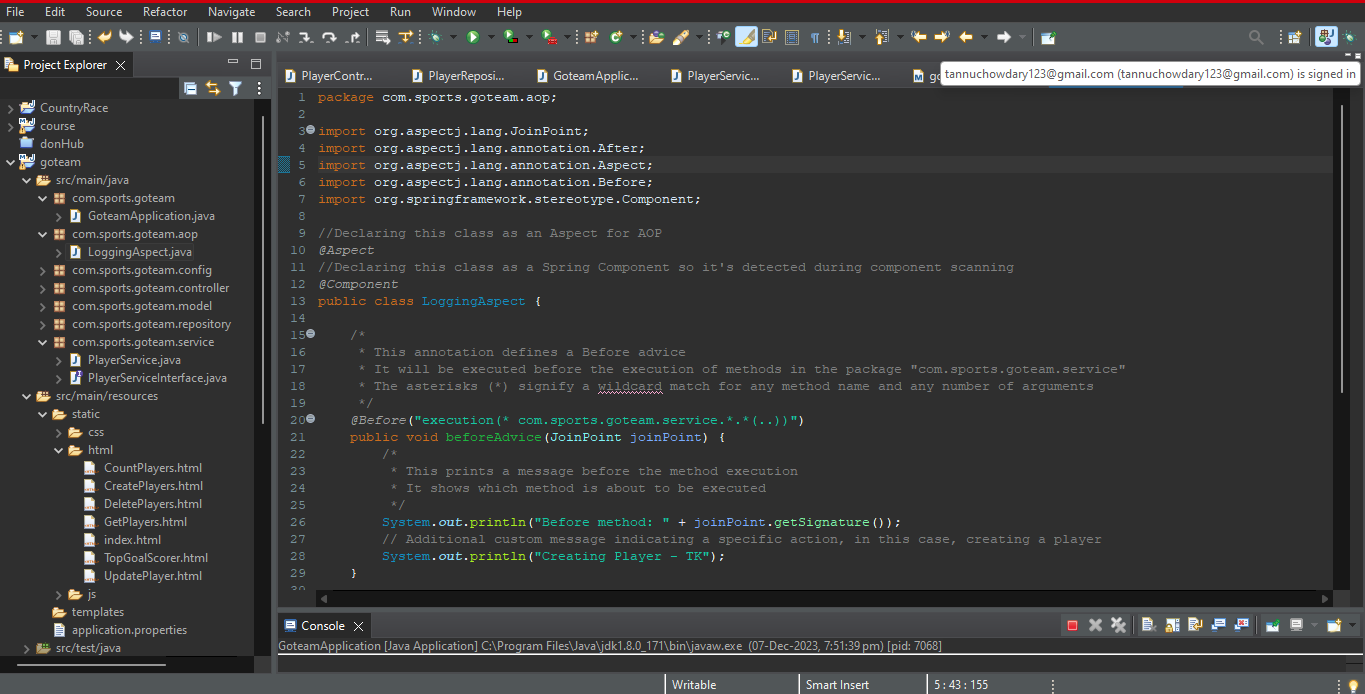
This approach is particularly useful in large and complex applications where cross-cutting concerns such as logging, transaction management, and security checks are prevalent. By handling these concerns separately, AOP contributes to a cleaner, more organized codebase.

**Steps done to achieve what is needed:**

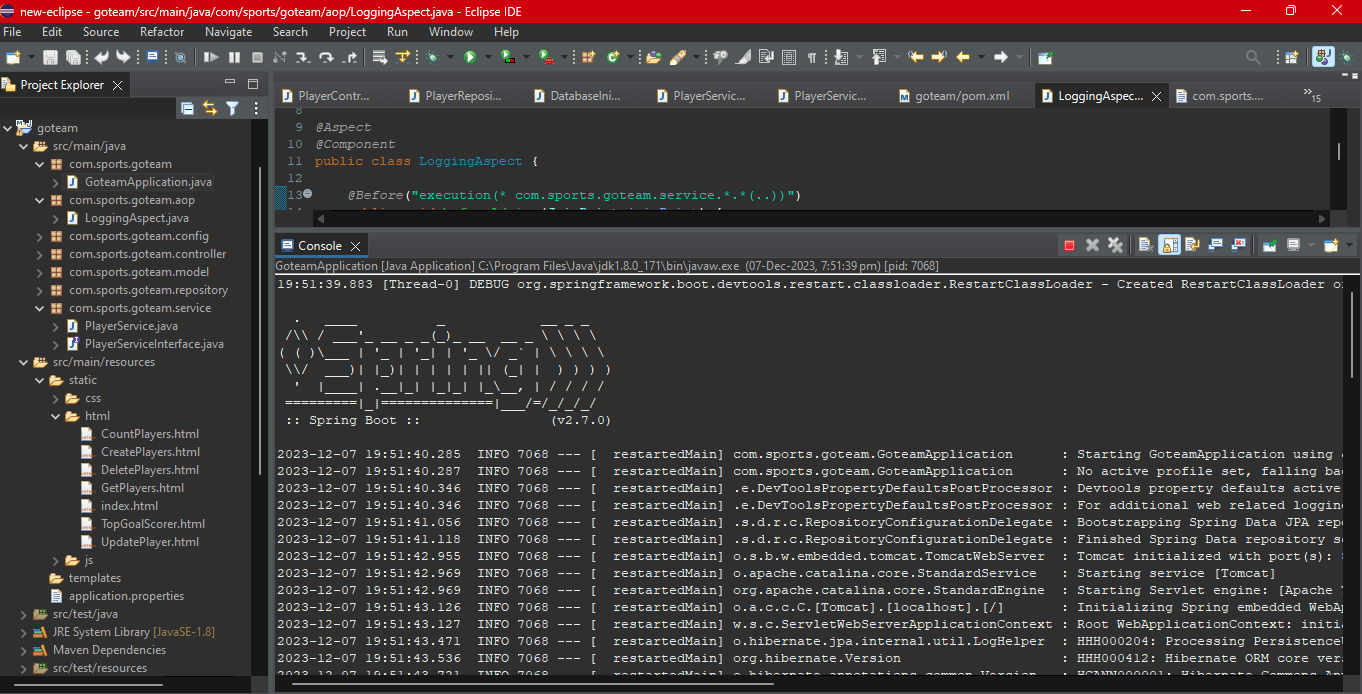
1. **Backend Configuration:**
2. **pom.xml:**

* **org.springframework.boot:spring-boot-starter-aop** enables us AOP capabilities in our Spring Boot project, allowing us to define and use aspects, join points, advices, and pointcuts in our application. ****

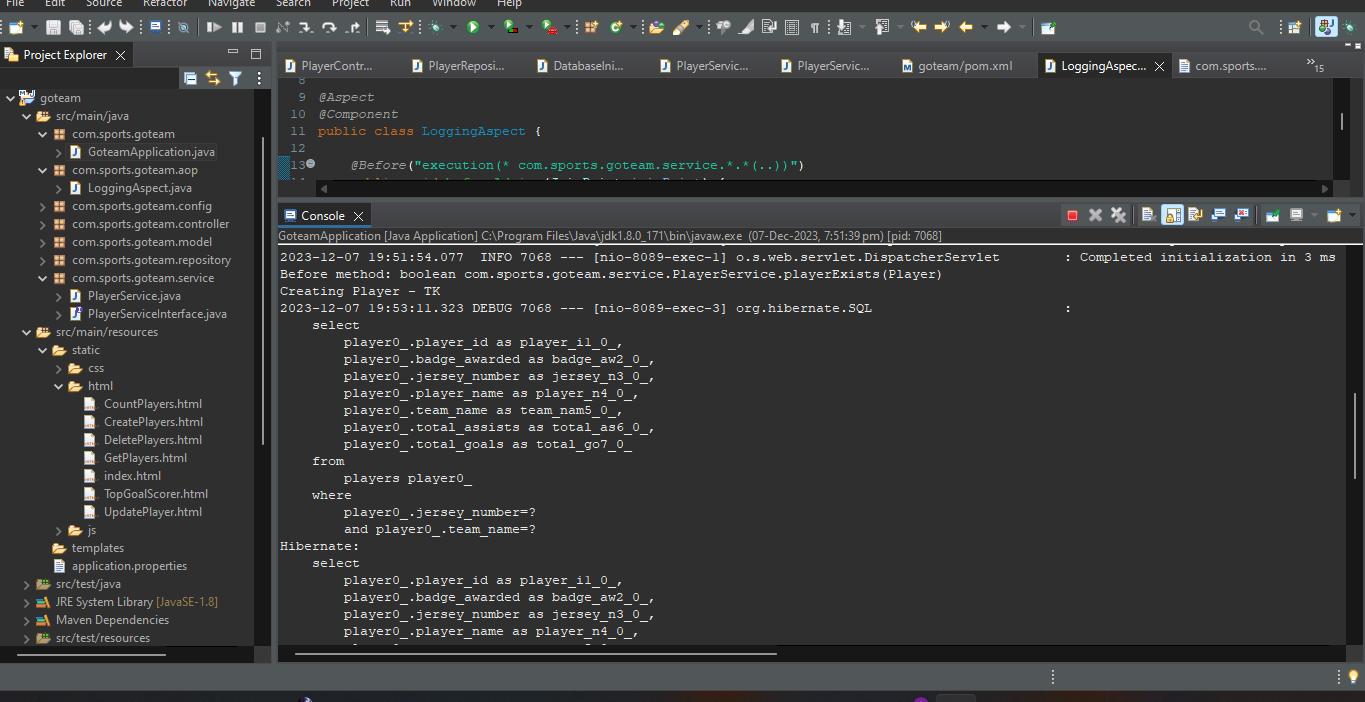
1. **Maven Lifecycle Management:** Throughout the development process, the project was cleaned to remove all files generated by the previous build, installed .jar files in the local repository, and updated to fetch any new dependencies.
2. **LoggingAspect.java:** This class LoggingAspect is an aspect in Spring AOP, which contains two advices (@Before and @After) that are executed before and after the methods of any class in the com.sports.goteam.service package. These advices are helpful for cross-cutting concerns like logging, where you can add functionality across various points of the application without modifying the main business logic.

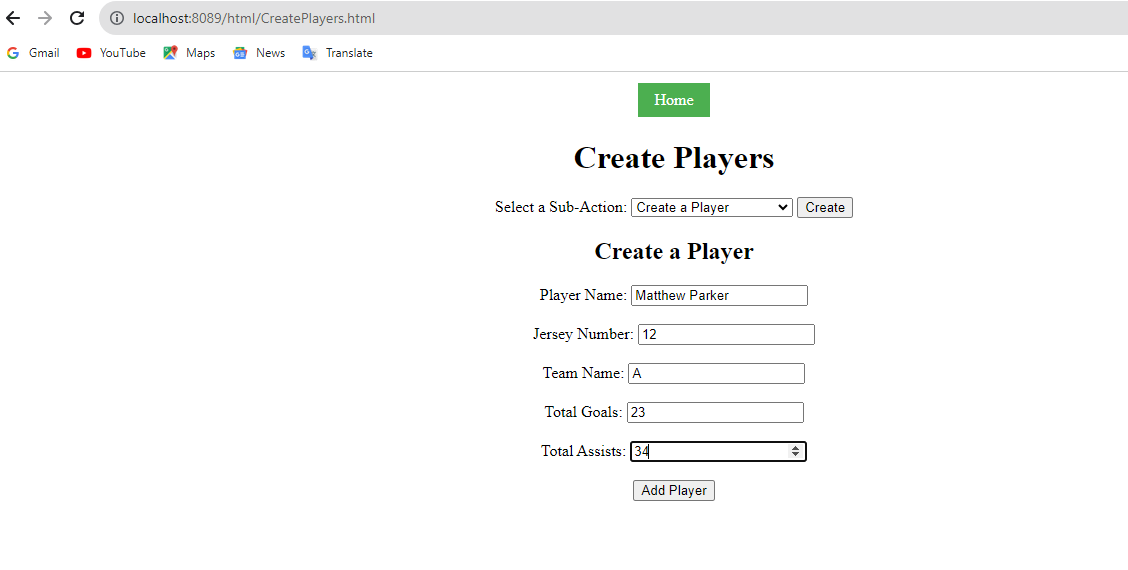
****

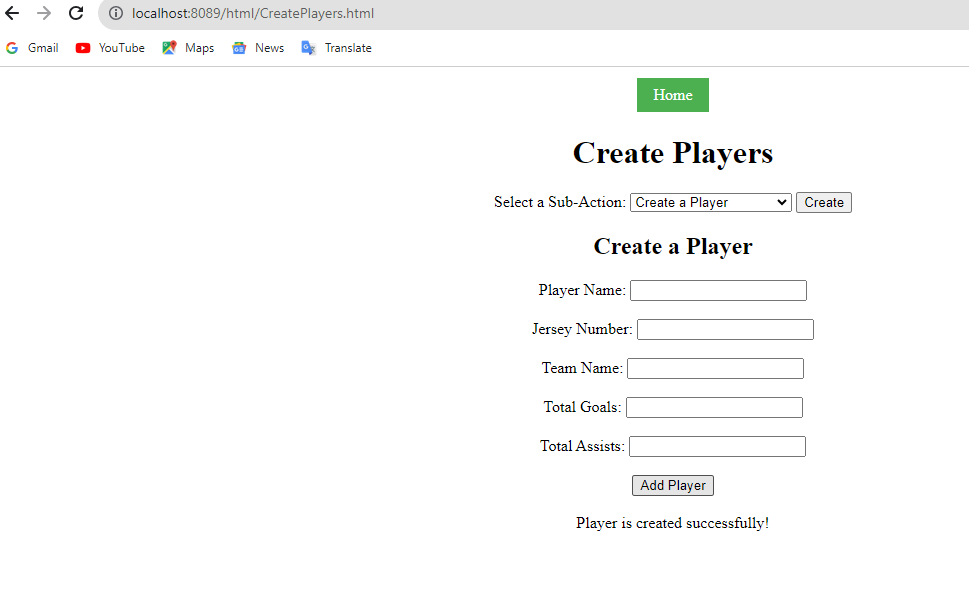
1. **Application Up and Running:**

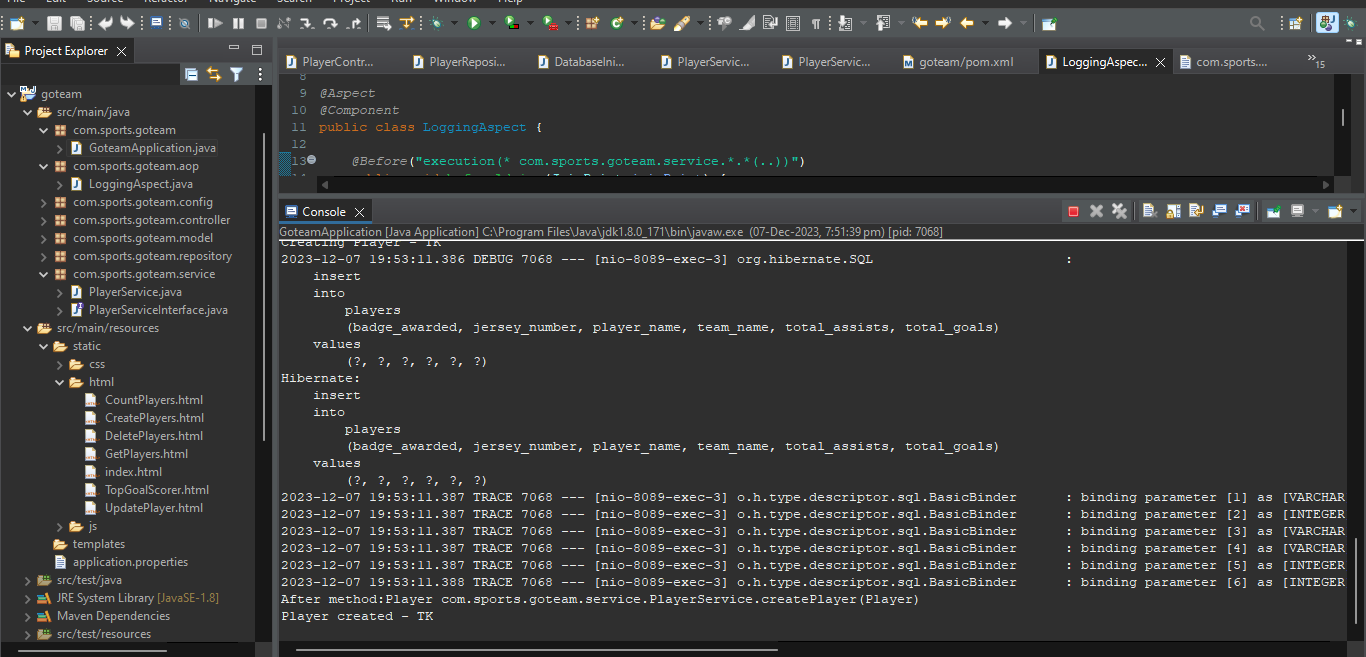
****

1. **Inserting players to the database via UI:**

****

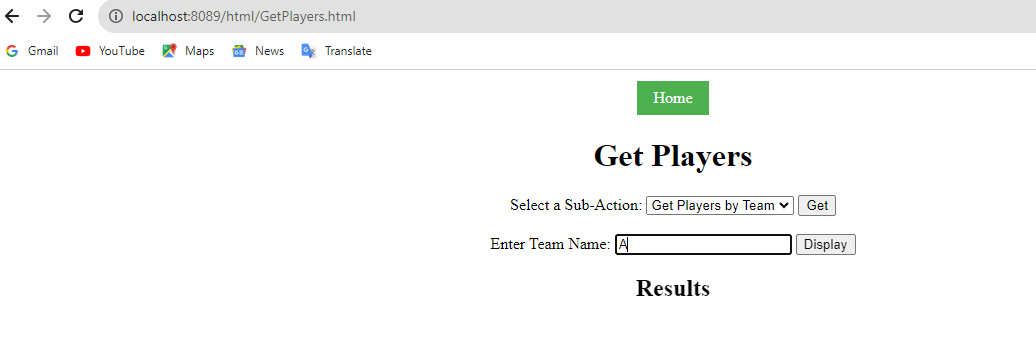
****

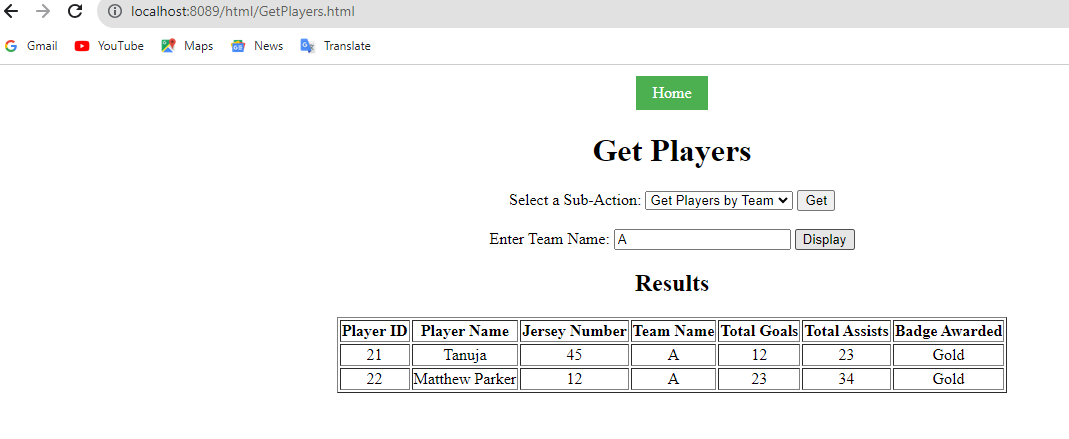
****

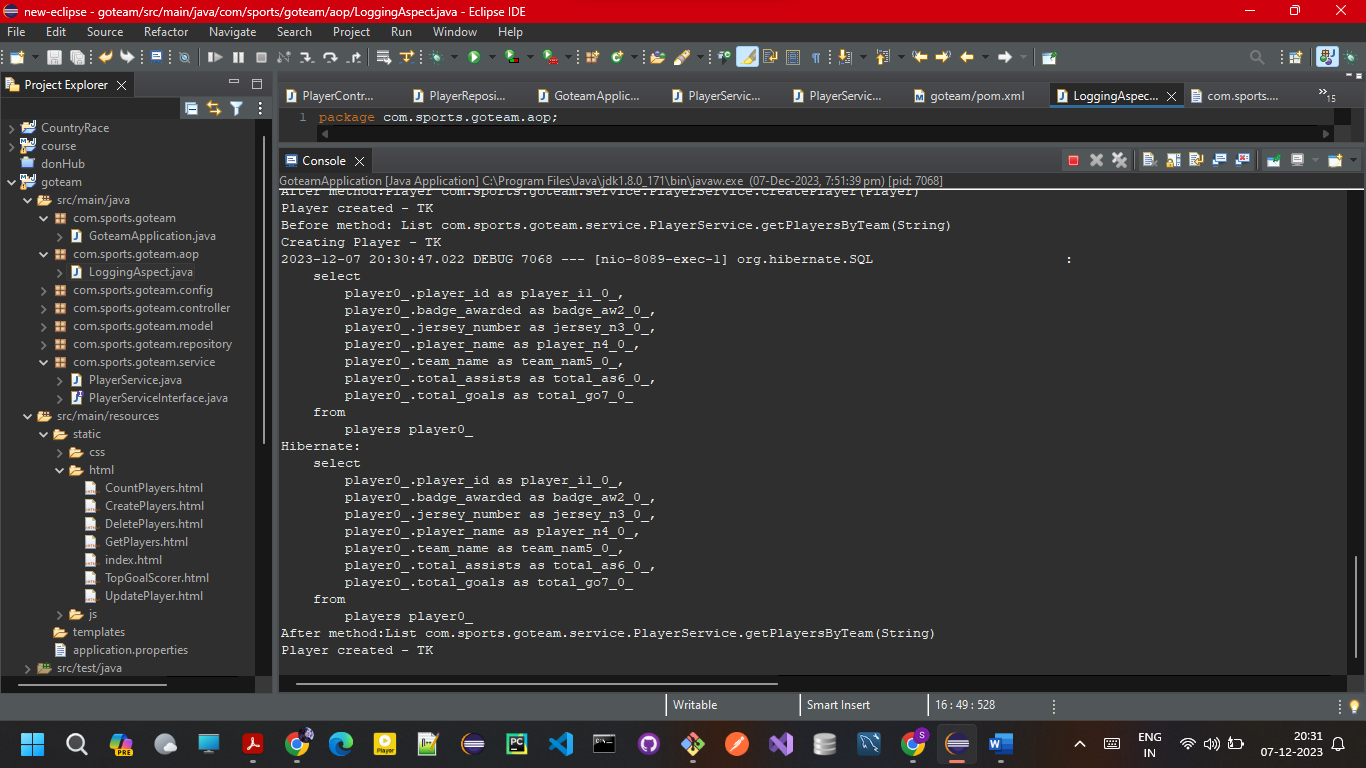
****

1. **Reading players from Sqlite:**

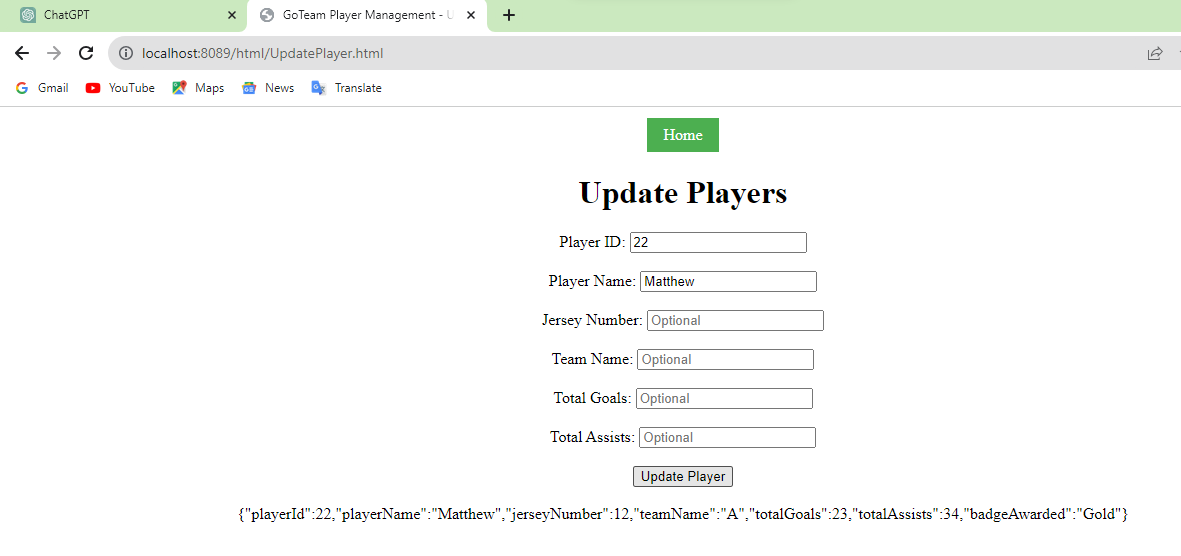
**Reading player by teamname:**

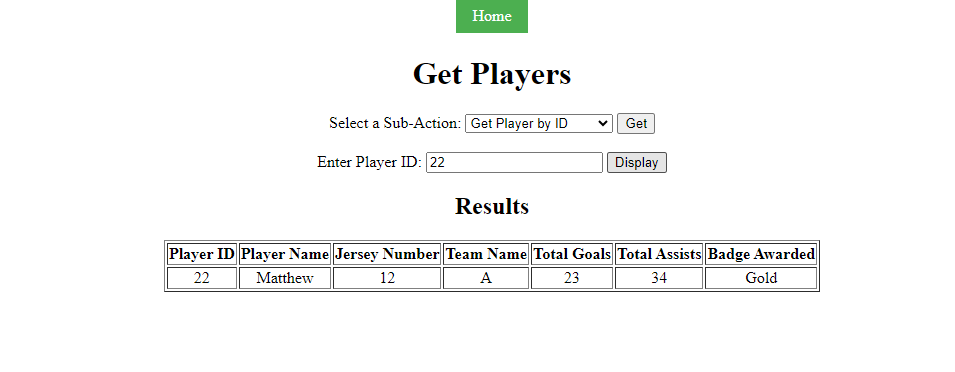
****

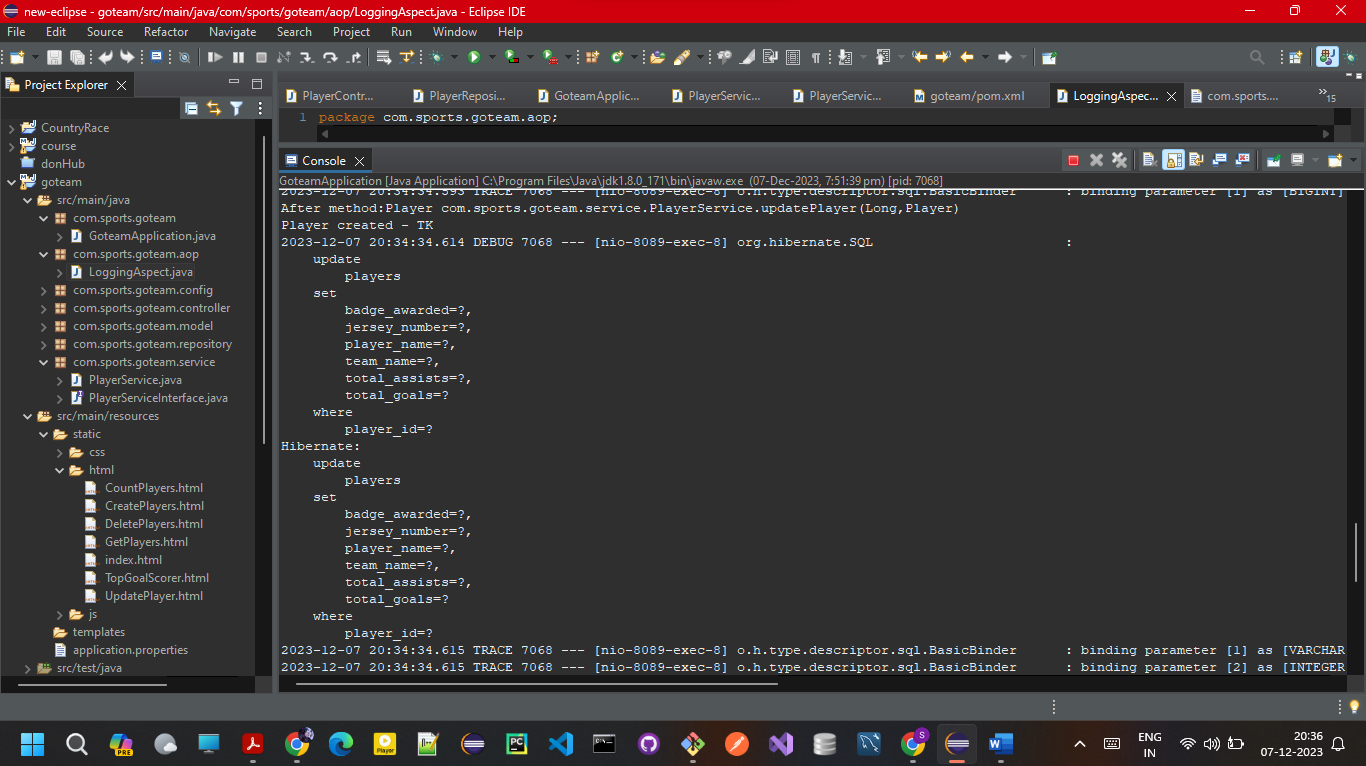
****

****

1. **Updating Players in the database via UI:**

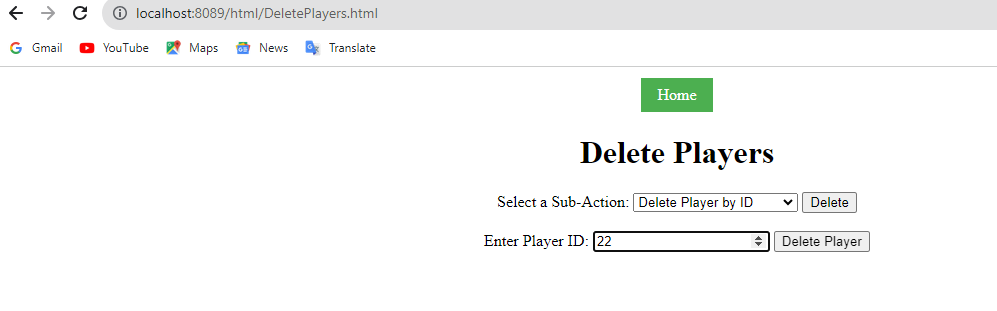
****

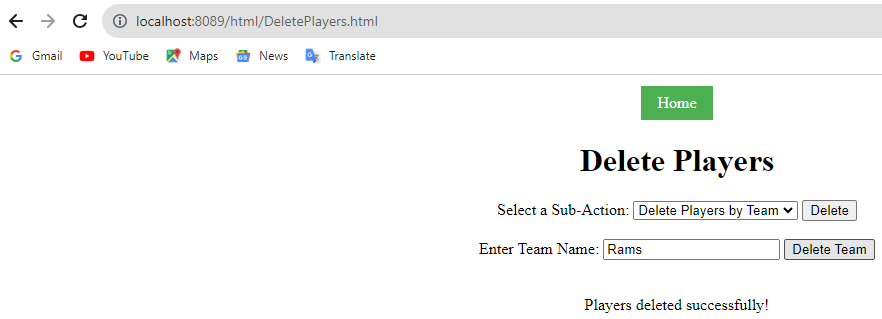
****

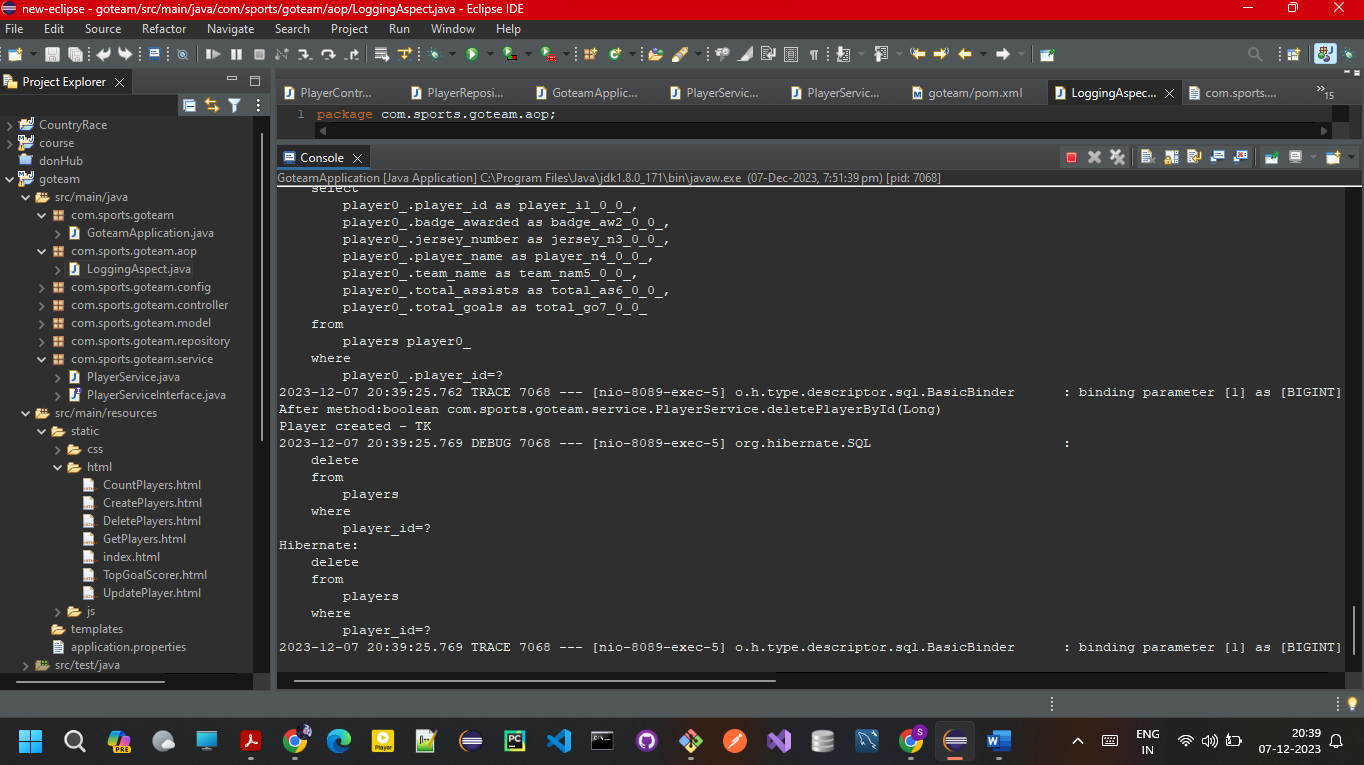
****

1. **Deleting players from Sqlite using UI:**

**Deleting player by id:**

****

****

****